

CSC 7003 : Basics of Software Engineering

J Paul Gibson, A207

`paul.gibson@int-edu.eu`

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/>

Version Control

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/L8-VersionControl.pdf>

Version control : background

Version control is also known as *resource control* or *source control*

It is the management of changes to documents, programs, and other information stored as computer files.

It is mostly used in software development, where a team of people may change the same files, and it is an important part of software configuration management

Changes – known as revisions - are usually identified by a unique ID - the *revision number*,

Each revision is usually associated with a timestamp and the person making the change.

Revisions can be compared, restored, and with some types of files, merged.

Version control : a (selected) history

Local

- 1972 SCCS
- 1982 RCS

Client-Server

- 1990 CVS
- 2000 Subversion

Distributed

- 2001 GNU arch
- 2000 DCVS
- 2003 SVK
- 2005 Bazaar
- 2007 Fossil

Distributed Systems

1. No canonical, reference copy of the code base exists by default; only working copies.
2. Common operations such as commits, viewing history, and reverting changes are fast, because there is no need to communicate with a central server.
3. Each working copy is effectively a remote backup of the code base and change history, providing natural security against data loss.

Version control : some key articles

The Source Code Control System , Marc J Rochkind, 1975

Design, implementation, and evaluation of a Revision Control System , Walter F Tichy, 1982

On Optimistic Methods for Concurrency Control , H.T. Kung and John T. Robinson, 1981

Version control : why?

Reversion: If you make a change, and discover it's not viable, how can you revert to a code version that is known good?

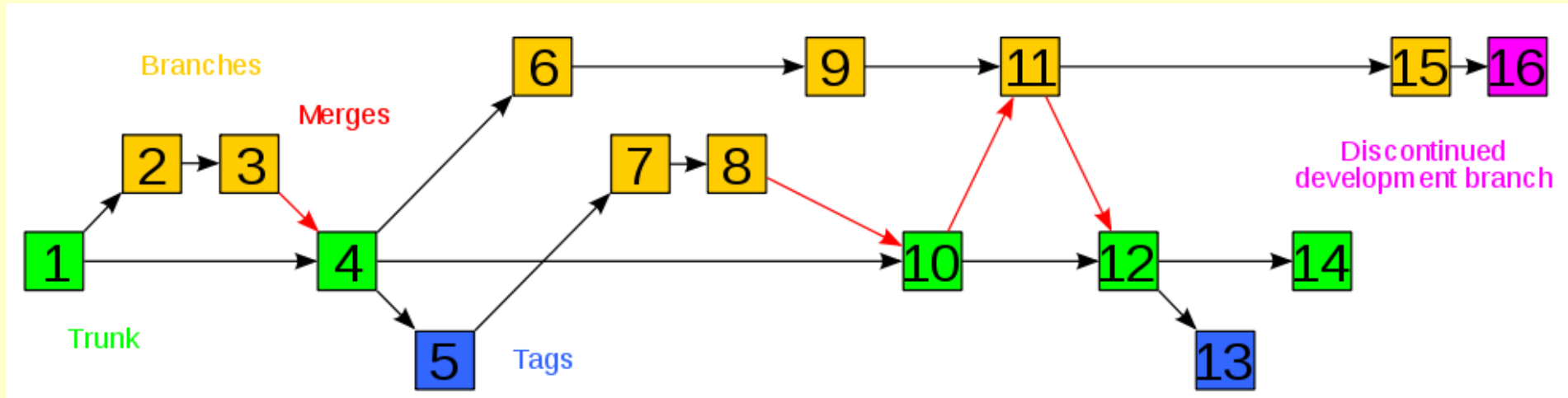
Change/Bug Tracking: You know your code has changed; do you know who, when and why? (When and where the new bug was introduced?)

Branches: How to introduce a completely new feature or concept and not mess up the working code?

Merging branches: If I divide up the code, how to merge new code with old code?

Parallel Development: How to manage independent developers making different changes to the same code?

Version Control: fundamental concepts



Tags (Baselines/Labels) – important snapshot of a project

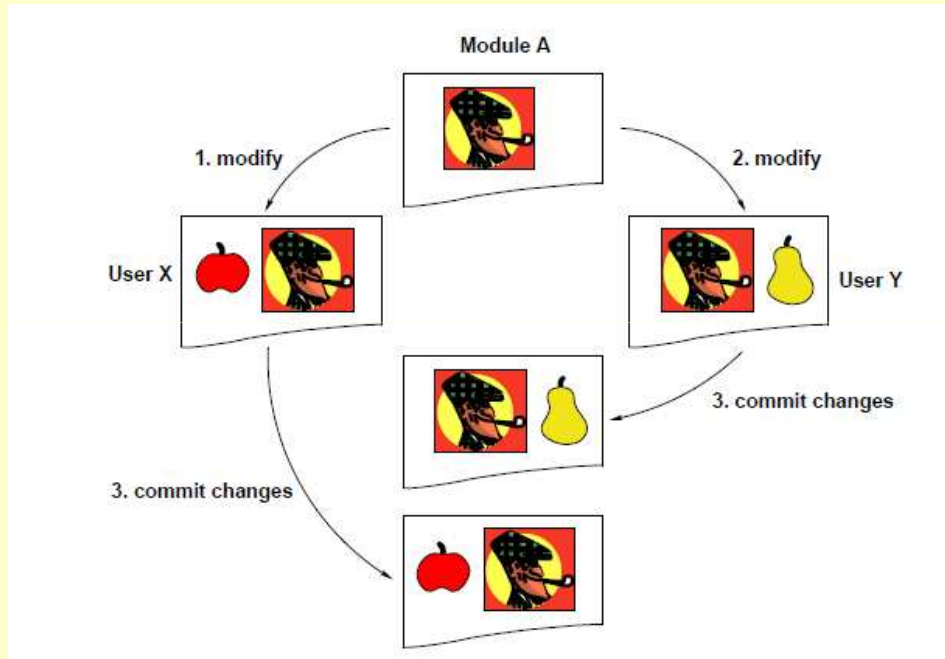
Branch - two (or more) copies of a project that may develop at different speeds or in different ways independently of each other.

Trunk (Baseline/Mainline) - The unique line of development that is not a branch

Merge - an operation in which two sets of changes are applied to a file or set of files or branches.

Version Control: fundamental concepts

Parallel Development: How to manage independent developers making different changes to the same code?



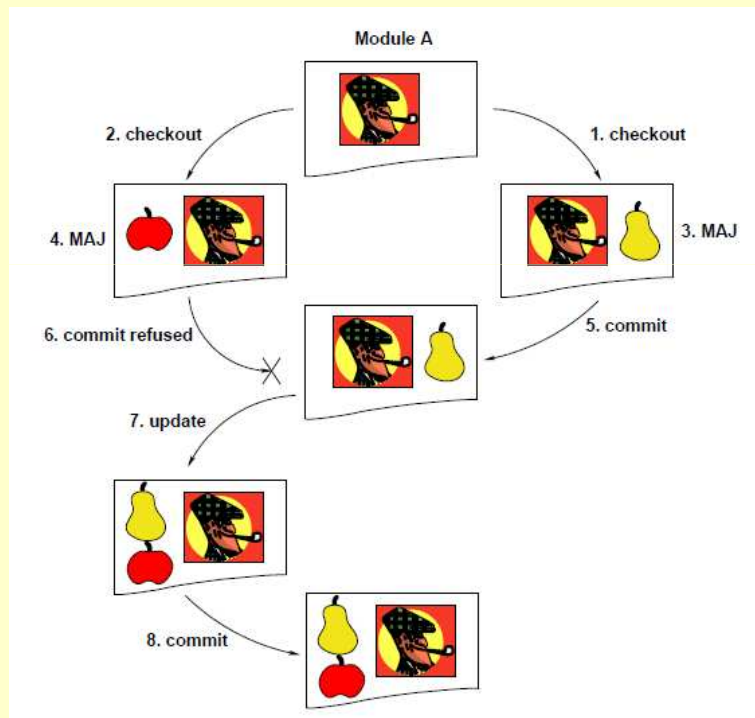
Solution: an *access protocol*

Most VCSs follow 1 of 2 approaches:

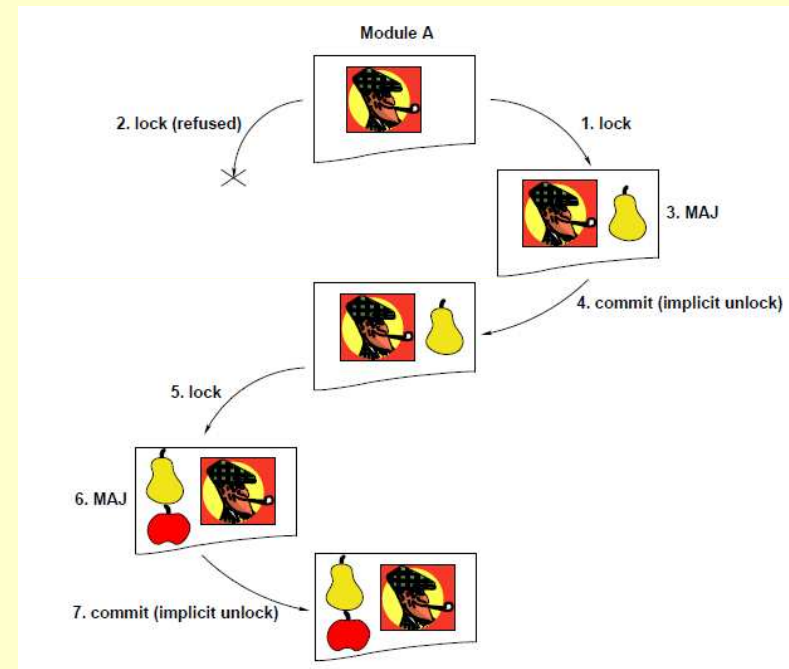
- Copy-Modify-Merge
- Lock-Modify-Unlock

Version Control: fundamental concepts

Parallel Development: How to manage independent developers making different changes to the same code?



Copy-Modify-Merge



Lock-Modify-Unlock

“Always Use Source Code Control”

From “The pragmatic programmer” by Andrew Hunt and David Thomas, 1999, an excellent advanced programming book:

“Always. Even if you are a single-person team on a one-week project. Even if it’s a “throw-away” prototype. Even if the stuff you’re working on isn’t source code. Make sure that everything is under the source code control — documentation, phone number list, memos to vendors, makefiles, build and release procedures, that little shell script that burns the CD master — everything. ... Even if we’re not working on a project, our day-to-day work is secured in a repository.”

Example Use Case Scenario with subversion

Installation

The first step for using Subversion is installing it. This depends on your system.

In Ubuntu and Debian, execute the following command:

```
$ sudo apt-get install subversion
```

Subversion has packages for many systems, Linux (Gentoo, Debian, Fedora, ...), Mac OS X, FreeBSD, Windows, ...

Subversion is already installed locally on your Fedora machines

It is most common to install subversion on a single machine (server) that is used by all members of the development team – you will have access to such a server for your projects

Administration: SVNadmin

The first Subversion tool we will use is **SVNadmin**, which is for administration tasks, like creating repositories, creating user accounts for repositories, making backup dumps, etc.

For example, to **create a local SVN referential** we execute the command:

```
SVNadmin create /home/gibson/Development/SVN
```

There will now be a new folder (SVN) in my Development folder.

I called this repository SVN, but you can call it whatever you like. (You can also put it anywhere you want). Subversion uses this directory to store information about your projects, like file revisions. You won't need to directly deal with this directory, so keep it safe and don't play around with it!

It is also a good idea to add an environment variable which provides a URL for this root directory, eg:

```
$ export SVN_HOME=file:///home/gibson/Development/SVN
```

Importing Projects

Now that we have a repository, we will use the SVN tool to import and manage projects.

To import a project, first create a directory for it in your (local) repository. To do so run SVN mkdir:

```
$ SVN mkdir $SVN_HOME/PROJECTNAME
```

(Replace the *PROJECTNAME* with the new project name, for example *CSC7003*)

Subversion will open your default text editor and ask you to enter a log message. Enter an explanation of what you're doing, save, and exit the editor.

You should see the message: **Committed revision 1.**

Importing Projects – a SampleProject example

Imagine that you have a `SampleProject` that already contains:

2 text files – `Document1.txt` and `Document2.txt`, and
a folder – `SubFolder1` – that contains a single text file `Document3.txt`

We wish to import this into our SVN repository, we execute the following at the root of our `SampleProject` directory to add to a `SampleProject` directory in SVN:

```
$ SVN mkdir $SVN_HOME/CSC7003/SampleProject  
Committed revision 2  
$ SVN import $SVN_HOME/CSC7003/SampleProject
```

```
Adding      SubFolder1  
Adding      SubFolder1/Document3.txt  
Adding      Document1.txt  
Adding      Document2.txt
```

Listing Projects – directories and files

```
$ SVN list $SVN_HOME
```

```
CSC7003/
```

```
$ SVN list $SVN_HOME/CSC7003/SampleProject
```

```
Document1.txt
```

```
Document2.txt
```

```
SubFolder1/
```

```
$ SVN list $SVN_HOME/CSC7003/SampleProject/SubFolder1
```

```
Document3.txt
```

Checking Out Projects

In a local directory where you wish to work on a local copy of the files, you can checkout the project from the repository:

```
$SVN checkout $SVN_HOME/CSC7003/SampleProject
```

```
A PROJECTNAME/SubFolder1
A PROJECTNAME/SubFolder1/Document3.txt
A PROJECTNAME/Document1.txt
A PROJECTNAME/Document2.txt
Checked out revision 3.
```

The 'A' is a flag to show that these were **added** in the latest revision

Decoding SVN output flags:

Decoding some SVN output

FLAG	meaning
U	Updated ... received changes from the server.
A	Added ... file/directroy was added (from the server).
D	Deleted ... file/directory was deleted from your sandbox. This means the file is no longer needed. Never fear, it is still recoverable if you need to look at it for some reason.
R	Replaced ... file/directory was deleted from the repository but replaced with something by the same name. Subversion considers them to be distinct.
G	merGed ... received changes from the server, but your local copy was modified (by you). The changes were either non-intersecting or exactly the same as the changes you made. "Good to Go"
C	Conflict ... received changes from the server, but your local copy was modified (by you). The changes overlap and are not identical. This will require human intervention to resolve.

The meaning of these will become clearer as we use SVN

Modify and status check.

In our local copy if we make a change to our local copy of Document1.txt , eg by adding the line « A first modification », we can verify that SVN has updated the status of the project correctly

```
$SVN status Document1.txt
```

```
M      Document1.txt
```

```
$SVN status --verbose Document1.txt
```

```
M      3      3 gibson      Document1.txt
```

```
$SVN status SampleProject
```

```
M      SampleProject/Document1.txt
```

Commit the change.

In the top level directory of our local copy of the project, we can commit our change to the SVN repository

`$SVN commit`

```
Sending      Document1.txt
Transmitting file data .
Committed revision 4.
```

Difference check.

```
$SVN diff -r 3:4 $SVN_HOME/CSC7003/SampleProject
```

Index: Document1.txt

=====

=

--- Document1.txt (revision 3)

+++ Document1.txt (revision 4)

@@ -1 +1,4 @@

A sample text document

+

+

+A first modification

Useful commands for reading from repository

checkout

status

log

diff

update

Useful commands for writing to the repository

add

rm

delete

rename

commit

Repository on a server (We can use, eg, the picoforge server at TMSP)

We can access the files on a server using the required protocol – [http://](#) or [https://](#) or [SVN+ssh://](#)

When specifying files, we no longer use [file:///](#) (which is for local access) but one of the other access protocols.

Anonymous SVN Access (on picoforge)

This project's SVN repository can be checked out through anonymous SVN (When prompted for a password for *guest*, simply put *guest* as password.)

```
$SVN co SVN+ssh://guest@picoforge.int-evry.fr/PROJECTNAME
```

Developer SVN Access via SSH

Only project developers can access the SVN tree via this method. SSH must be installed on your client machine. You will need an SVN password

```
$SVN co SVN+ssh://username@picoforge.int-evry.fr/CSC7003pg
```

We can usually follow the progress of a remotely hosted SVN project through a web interface. For example, at the picoforge web site:[http://picoforge.int-evry.fr/...](http://picoforge.int-evry.fr/)

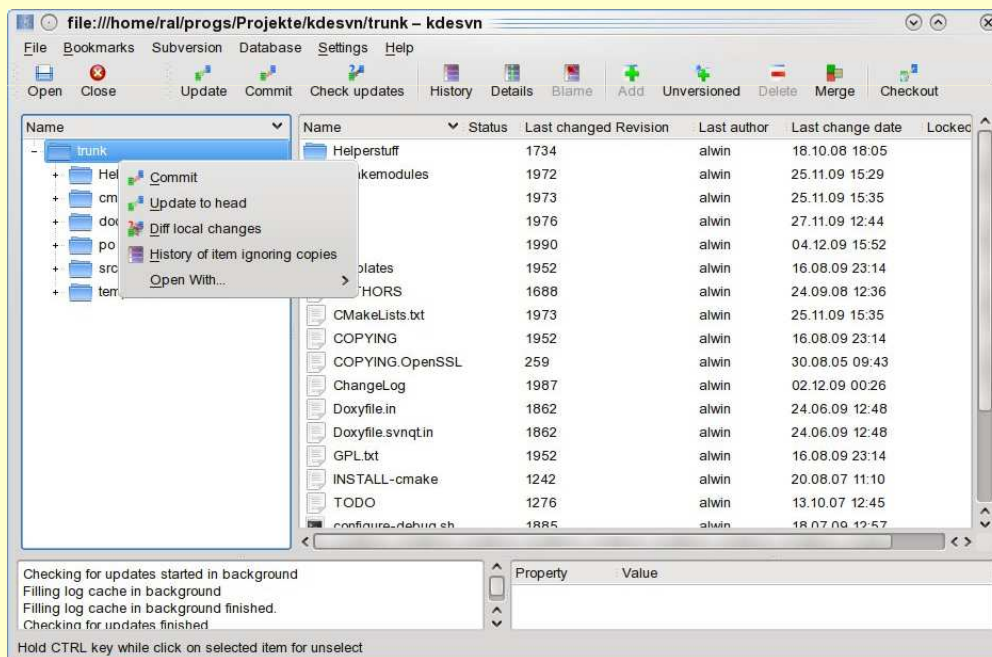
The screenshot shows a Mozilla Firefox browser window displaying the picoforge web interface. The page title is "[Project home] - Mozilla Firefox". The address bar shows the URL "https://picoforge.int-evry.fr/phpgroupware/picol". The page content includes a navigation menu on the left with links like "Project choice", "Project home", "CVS Access", "SVN Access", "Mailing lists", "Wikis", "Bug report", and "Project FAQ". The main content area displays the "Current Project: csc7003pg" and provides administrative links: "Administrate this project", "Project Home Page", and "Exit from Project". A table shows project details: "project description*" with ID 1290076952, "Short Title : csc7003pg", "Long Title : CSC7003 Software Engineering for MSC SAI students", "Short Description : CSC7003 Software Engineeri", "Long Description : Repository for MSc SAI students for joint project work in CSC7003 module", "Type : teaching", "Theme : programming", "Web Site Home Page : http%3A%2F%2Fpicoforge.int-evry.fr%2Fprojects%2F", "License : GNU General Public License (GPL)", and "Status : public". To the right, there are sections for "Developer list" (Admins: gibson, Developers:) and "External application list" (Web CVS access, Web SVN access, Project's website, Sources Snapshots). A "No task to do!" message is shown above a table with columns: "Todo id", "Access", "Project", "Urgency", "Status", "Summary", "Start Date", "End date", "Created By", "Category", "View", "Edit", "Delete". An "Add" button is below the table. The footer shows "gibson - Friday 2011/01/07" and "Powered by phpGroupWare version 0.9.16.011".

But, it is more convenient to install a SVN client GUI

Using a SVN graphical client:

There are many different clients available which provide a nice GUI front end to a SVN server (local or networked).

For Unix we recommend: kdeSVN (see <http://kdeSVN.alwins-world.de/>)



This is installed and configured on your Fedora machines.

There is a handbook at:

<http://kdeSVN.alwins-world.de/extras/book/>

Using a VCS in conjunction with an IDE (like Eclipse)

We recommend the Subclipse Plugin for Eclipse/Subversion Integration:

`http://subclipse.tigris.org/`

A simple tutorial for installation can be found at:

`http://wheelersoftware.com/articles/install-subclipse-eclipse-svn.html`

A simple tutorial for checking out projects can be found at:

`http://wheelersoftware.com/articles/eclipse-svn-project-checkout.html`